Chapter - 5: Appendix

Import the Dataset and Necessary Libraries

In [ ]:
```
import pandas as pd
from tabulate import tabulate
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Specify the path to the Excel file
excel_file_path = "CDM_nfca.xlsx"

# Read the Excel file into a DataFrame
df = pd.DataFrame(pd.read_excel(excel_file_path))

# Display the first few rows of the dataframe
df.head(10)
```

index	Title	Theme	Subject
0	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Amusement rides -- Dodgems -- Or and Spooner
1	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Transport
2	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Transport
3	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Amusement rides -- Ark -- Lakin
4	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Amusement rides -- Dodgems

Primary Investigation

In [ ]:

```
# Display the shape of the dataset
print(df.shape)

# Display the information of the dataframe
df.info()
```

```
(76676, 44)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76676 entries, 0 to 76675
Data columns (total 44 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Title                        76676 non-null  object
 1   Theme                        76387 non-null  object
 2   Subject                      76630 non-null  object
 3   Name                         61391 non-null  object
 4   Description                  76663 non-null  object
 5   Type                         76657 non-null  object
 6   Accession Number             76663 non-null  object
 7   Derivative                   17341 non-null  object
 8   Digital Master               76406 non-null  object
 9   Place Name                   64636 non-null  object
 10  Place Name (specific)        75991 non-null  object
 11  Geotag Location              56965 non-null  object
 12  Date of Subject              76019 non-null  object
 13  Period                       76507 non-null  object
 14  Creator                      75913 non-null  object
 15  Publisher                    76666 non-null  object
 16  Donor                        75181 non-null  object
 17  Collection Number            76435 non-null  object
 18  Collection Name              76537 non-null  object
 19  Digitiser                    76484 non-null  object
 20  Rights                       76666 non-null  object
 21  Permitted Uses               76666 non-null  object
 22  Contact for Further Uses     76666 non-null  object
 23  Date Created                 75803 non-null  object
 24  Date Digitised               76601 non-null  object
 25  Original Format (specific)   76554 non-null  object
 26  Original Format (general)    76666 non-null  object
 27  Physical Extent              52837 non-null  object
 28  Orientation                  76432 non-null  object
 29  Digital Format               76529 non-null  object
 30  Digital Extent               76007 non-null  object
 31  Digital Size                 76579 non-null  object
 32  Colour                       75828 non-null  object
 33  Sound                        7 non-null      object
 34  Quality                      344 non-null    object
 35  Identifier                   76582 non-null  object
 36  Language                     920 non-null    object
 37  OCLC number                  0 non-null      float64
 38  Date created                 76676 non-null  datetime64[ns]
 39  Date modified                76676 non-null  datetime64[ns]
 40  Reference URL                76676 non-null  object
 41  CONTENTdm number             76676 non-null  int64
 42  CONTENTdm file name          76676 non-null  object
 43  CONTENTdm file path          76676 non-null  object
dtypes: datetime64[ns](2), float64(1), int64(1), object(40)
memory usage: 25.7+ MB
```

In [ ]:

```
## MISSING VALUES

# Count the number of missing values in each column
```

```
missing_counts = df.isna().sum()
missing_percentile = (df.isna().sum()/76676) * 100

# Display the missing value counts
missing_data = pd.DataFrame({
    'Missing Count': missing_counts,
    'Missing Percentage': missing_percentile
})
missing_data
```

index	Missing Count	Missing Percentage
Title	0	0
Theme	289	0.37691063696593463
Subject	46	0.05999269654129063
Name	15285	19.93452970942668
Description	13	0.01695445771819083
Type	19	0.024779592049663516
Accession Number	13	0.01695445771819083
Derivative	59335	77.38405759298868
Digital Master	270	0.35213104491627106
Place Name	12040	15.702436225155198

Page **1 of 5** | Go to page: 1 Show 10

Data Pre-processing

In [ ]:

```
## Treating Missing Values
# Remove missing values in columns which have less than 5% of missing values
# Filter out the columns where the missing percentage is less than 5%
columns_to_treat = missing_data[missing_percentile < 5].index
df = df.dropna(subset=columns_to_treat)

# Display the treated  missing value counts and percentages
missing_counts = df.isna().sum()
missing_percentile = df.isna().sum()/76676 * 100
missing_data = pd.DataFrame({
    'Missing Count': missing_counts,
    'Missing Percentage': missing_percentile
})
missing_data
```

index	Missing Count	Missing Percentage
Title	0	0
Theme	0	0
Subject	0	0
Name	14462	18.861182116959675
Description	0	0
Type	0	0
Accession Number	0	0
Derivative	56263	73.37758881527466
Digital Master	0	0
Place Name	11163	14.558662423704941

Page **1 of 5** | Go to page: 1 Show 10

In [ ]:

```
# Get the list of distinct themes
distinct_themes = df['Theme'].unique()

# Print the distinct themes after
print("Distinct Themes")
for theme in distinct_themes:
    print(theme)

print("Number of Unique Themes: "+ str(len(distinct_themes)))
```

Distinct Themes
Fairs
Amusement parks
Circus
Steam-engines -- Exhibitions
Fetes
Fairs
Great Ouse River (England)
Church buildings
Exhibitions
Machinery models
Factories
Museums
Zoos
Fairgrounds
Seaside Entertainment
Factories; Fairs
Amusement Parks
Museums; Steam-engines -- Exhibitions
Cemeteries
Road rollers
Weddings
Parks
Seaside resorts
Rivers
Traction-engines
Buses
Fairs; Exhibitions
Fairs; Exhibitions; Variety and Music Hall
Motion picture theaters
Variety and Music Hall; Exhibitions
Steam-engines -- Exhibitions; Fairs; Exhibitions
Bingo
Ploughing
Taverns (Inns)
Plowing
Castles
Narrow gauge railroads
Cafeterias
Amusement parks; Zoos
Aquariums
Circus; Amusement parks
Circus; Exhibitions
Circus; Zoos
Zoos; Circus
Music-halls
Circus; Seaside Entertainment
Speedway motorcycle racing
Exhibition
fairs
Fairs; Circus
Exhibitions; Variety and Music Hall
Circus; Fair
Fairs; Amusement parks
Amusement parks
Number of Unique Themes: 54

In [ ]:
```
## Mapping Redundant and Inconsistent values in ‘Theme’
themes_mapping = {
    'Fetes' : 'Fairs',
    'Fairs ' : 'Fairs',
    'Fairs; Traction-engines; Steam-engines -- Exhibitions; Steam-engines;':
    'Fairs; Steam-engines -- Exhibitions': 'Fairs',
    'Fairs; Amusement Parks and Theme Parks; Fixed Entertainment Venue':'Fai
    'Fairs; Amusement parks': 'Fairs',
    'fairs':'Fairs',
    'Fairs; Circus':'Fairs',
    'Fairs; Circus; Exhibitions':'Fairs',
    'Fairs; Exhibitions':'Fairs',
    'Fairs; Exhibitions; Variety and Music Hall':'Fairs',
    'Circus; Amusement parks': 'Circus',
    'Circus; Exhibitions': 'Circus',
    'Circus; Zoos' : 'Circus',
    'Zoos; Circus' : 'Circus',
    'Fixed Entertainment Venue; Circus;':'Circus',
    'Circus; Seaside Entertainment': 'Circus',
    'Circus; Fair':'Circus',
    'Circus; Music Hall and Variety':'Circus',
    'Circus;':'Circus',
    'Circus; Variety and Music Hall;' : 'Circus',
    'Circus; Fairs;': 'Circus',
    'Circus; Zoos;': 'Circus',
    'Circus; Variety and Music Hall':'Circus',
    'Seaside Resorts (Seaside Entertainment); Performers': 'Seaside Resorts
    'Seaside Resorts (Seaside Entertainment): Performers': 'Seaside Resorts
    'Seaside Resorts (Seaside Entertainment): Amusement Parks':'Seaside Reso
    'Great Ouse River (England)':'Seaside Resorts (Seaside Entertainment)',
    'Seaside Entertainment':'Seaside Resorts (Seaside Entertainment)',
    'Rivers' :'Seaside Resorts (Seaside Entertainment)',
    'Seaside Resorts (Seaside Entertainment)':'Seaside Resorts (Seaside Ente
    'Seaside resorts':'Seaside Resorts (Seaside Entertainment)',
    'Variety and Music Hall; Circus;':'Variety and Music Hall',
    'Variety and Music Hall; Circus; Exhibitions' : 'Variety and Music Hall'
    'Amusement Parks and Theme Parks;': 'Amusement Parks',
    'Amusement Parks and Theme Parks; Seaside Entertainment' : 'Amusement Pa
    'Amusement parks; Zoos': 'Amusement Parks',
    'Amusement parks; Zoos ': 'Amusement Parks',
    'Amusement parks ':'Amusement Parks',
    'Amusement parks':'Amusement Parks',
    'Amusement Parks and Theme Parks; Circus':'Amusement Parks',
    'Exhibitions; Variety and Music Hall' : 'Exhibition',
    'Exhibitions; Circus; Variety and Music Hall' : 'Exhibition',
    'Exhibition; Variety and Music Hall' : 'Exhibition',
    'Exhibitions; Circus':'Exhibition',
    'Exhibition; Variety and Music Hall; Circus':'Exhibition',
    'Exhibitions; Fairs':'Exhibition',
    'Performers; Circus':'Performers',
    'Showpeople':'Performers',
    'Performer':'Performers',
    'Performers; Seaside Entertainment':'Performers',
    'Peterborough Cathedral':'Church buildings',
    'Cemeteries':'Church buildings',
```

```
    'Fairground':'Exhibitions',
    'Music Hall and Variety':'Exhibitions',
    'Fairgrounds; Exhibitions':'Exhibitions',
    'Fixed Enertainment Venue':'Exhibitions',
    'Traction-engines; Steam-engines -- Exhibitions; Steam-engines;':'Exhibi
    'Fixed Entertainment Venue;':'Exhibitions',
    'Variety and Music Hall;':'Exhibitions',
    'Steam-engines -- Exhibitions; Circus':'Exhibitions',
    'Speedway motorcycle racing':'Exhibitions',
    'Exhibition':'Exhibitions',
    'Steam-engines -- Exhibitions ':'Exhibitions',
    'NFA Logo opacity 30':'Exhibitions',
    'Exhibition; Circus':'Exhibitions',
    'Machinery models':'Exhibitions',
    'Steam-engines':'Exhibitions',
    'Museums; Steam-engines -- Exhibitions':'Exhibitions',
    'Variety and Music Hall; Exhibitions':'Exhibitions',
    'Steam-engines -- Exhibitions; Fairs; Exhibitions':'Exhibitions',
    'Variety and Music Hall':'Exhibitions',
    'Traction-engines':'Exhibitions',
    'Buses':'Exhibitions',
    'Motion picture theaters':'Exhibitions',
    'Steam-engines -- Exhibitions':'Exhibitions',
    'Narrow gauge railroads':'Exhibitions',
    'Plowing':'Ploughing',
    'Factories; Fairs':'Factories',
    'Fairgrounds':'Fairs'
}

# Apply the mapping to the 'Theme' column
df['Theme'] = df['Theme'].replace(themes_mapping)

# Get the updated list of distinct themes after grouping
distinct_themes = df['Theme'].unique()

print("Number of Unique Themes: "+ str(len(distinct_themes)))
```

```
Number of Unique Themes: 20
```

In [ ]:

```
# Extract individual years from the 'Period' column to create a new 'Year' c
df['Year'] = df['Period'].str.extract(r'(\d{4})')
```

In [ ]:

```
# install NLTK package
!pip install nltk
```

```
Collecting nltk
  Downloading nltk-3.8.1-py3-none-any.whl (1.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 45.9 MB/s eta 0:00:
00
Requirement already satisfied: regex>=2021.8.3 in /opt/conda/lib/python3.9/s
ite-packages (from nltk) (2023.5.5)
Requirement already satisfied: joblib in /opt/conda/lib/python3.9/site-packa
ges (from nltk) (1.2.0)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.9/site-package
s (from nltk) (4.65.0)
Requirement already satisfied: click in /opt/conda/lib/python3.9/site-packag
es (from nltk) (8.1.3)
Installing collected packages: nltk
Successfully installed nltk-3.8.1
```

In [ ]:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string

# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('averaged_perceptron_tagger')

# Load the set of English stopwords
stop_words = set(stopwords.words('english'))

# Function to remove named entities from text
def remove_named_entities(text):
    tokens = word_tokenize(text)
    tagged_tokens = nltk.pos_tag(tokens)
    non_entities = [word for word, tag in tagged_tokens if not is_named_enti
    return ' '.join(non_entities)

# Function to check if a POS tag represents a named entity
def is_named_entity(tag):
    return tag in ['NNP', 'NNPS']

# Function to tokenize a text and remove stop words, dates, punctuation mark
def tokenize_text(text):
    words = word_tokenize(text)
    words = [word.lower() for word in words]
    words = [word for word in words if word not in string.punctuation]
    words = [word for word in words if word not in stop_words]
    words = [word for word in words if not
             any(char.isdigit() for char in word) and
              word != '-' and word != "'s" and
              word != "photographed" and
              word != "photograph" and
              word != "photograph " and
              word != "digitisation" and
```

```
            word != "number" and
            word != "registration" and
            word != '--'  and
            word != '_' ]
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]
    return words

# Combine 'Description', 'Subject' and 'Theme' columns
df['tags'] = df['Description'].fillna('') + ' ' + df['Subject'].fillna('') +

# Remove named entities from 'tags' column
df['tags'] = df['tags'].apply(remove_named_entities)

# Tokenize the cleaned text
df['tags'] = df['tags'].apply(tokenize_text)

# Display the DataFrame with the 'tags' column containing the cleaned and to
df.head()
```

```
[nltk_data] Downloading package punkt to /home/noteable/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/noteable/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/noteable/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /home/noteable/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /home/noteable/nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/noteable/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

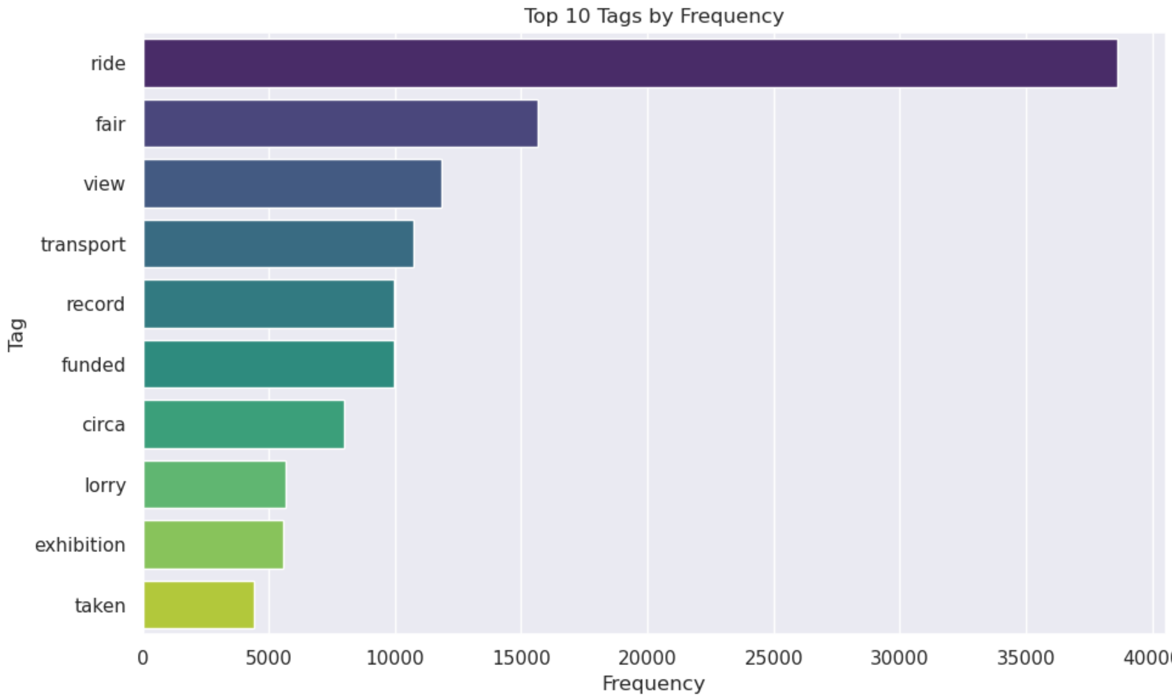
index	Title	Theme	Subject
0	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Amusement rides -- Dodgems -- Or and Spooner
1	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Transport
2	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Transport
3	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Amusement rides -- Ark -- Lakin
4	Jack Leeson photograph, Nottingham Goose Fair, 1958.	Fairs	Amusement rides -- Dodgems

Exploratory Data Analysis(EDA)

In [ ]:

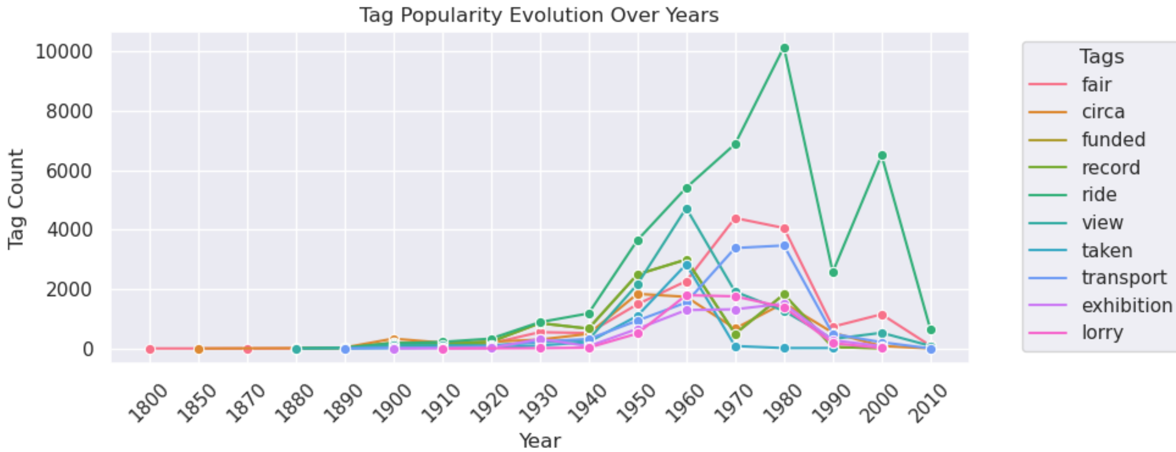
```
# Top 10 Tags by Frequency
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter

tags_data = df['tags'].tolist()

# Flatten the list of tokens into a single list
all_tags = [tag for tags in tags_data for tag in tags]

# Compute the frequency of each tag
tag_frequencies = Counter(all_tags)

# Convert the tag frequencies to a DataFrame for better visualization
tag_frequency_df = pd.DataFrame(tag_frequencies.items(), columns=['Tag', 'Fr

# Sort the DataFrame in descending order based on tag frequency
tag_frequency_df = tag_frequency_df.sort_values(by='Frequency', ascending=Fa

# Set up the plot using seaborn
plt.figure(figsize=(10, 6))
sns.barplot(x='Frequency', y='Tag', data=tag_frequency_df.head(10), palette=
sns.set(rc={"figure.figsize":(2, 4)})
# Customise the plot
```

```
plt.xlabel('Frequency')
plt.ylabel('Tag')
plt.title('Top 10 Tags by Frequency')
plt.tight_layout()

# Show the plot
plt.show()
```



In [ ]:

```
# Tag Popularity Evolution Over Years
# Select a subset of tags for visualisation (e.g., top 10 most common tags)
top_tags = tag_count_by_year.groupby('tags')['tag_count'].sum().nlargest(10)
top_tags_data = tag_count_by_year[tag_count_by_year['tags'].isin(top_tags)]

sns.lineplot(x='Year', y='tag_count', hue='tags', data=top_tags_data, marker
sns.set(rc={"figure.figsize":(10, 5)}) #width=3, #height=4
plt.xlabel('Year')
plt.ylabel('Tag Count')
plt.title('Tag Popularity Evolution Over Years')
plt.xticks(rotation=45)
plt.legend(title='Tags', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



In [ ]:
```
!pip install wordcloud
```

```
Collecting wordcloud
  Downloading wordcloud-1.9.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl (460 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 460.4/460.4 kB 17.8 MB/s eta 0:0
0:00
Requirement already satisfied: pillow in /opt/conda/lib/python3.9/site-packa
ges (from wordcloud) (9.2.0)
Requirement already satisfied: numpy>=1.6.1 in /opt/conda/lib/python3.9/site
-packages (from wordcloud) (1.23.5)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.9/site-p
ackages (from wordcloud) (3.6.0)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python
3.9/site-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.9/
site-packages (from matplotlib->wordcloud) (3.0.9)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.9/
site-packages (from matplotlib->wordcloud) (1.0.5)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.9/site
-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.
9/site-packages (from matplotlib->wordcloud) (4.37.4)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.
9/site-packages (from matplotlib->wordcloud) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.9/s
ite-packages (from matplotlib->wordcloud) (23.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.9/site-pac
kages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
Installing collected packages: wordcloud
Successfully installed wordcloud-1.9.2
```

In [ ]:
```
from wordcloud import WordCloud

# Group by 'Theme' and count the frequency of each theme
theme_counts = df['Theme'].value_counts()

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white').gener

# Display the word cloud using matplotlib
```

```
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Popular Themes')
plt.show()
```

In [ ]:
```
# Calculate the frequency of each theme
theme_freq = df['Theme'].value_counts()

# Create a new DataFrame to store the frequency table
theme_freq_table = pd.DataFrame({'Theme': theme_freq.index, 'Frequency': the

# Display the frequency table
theme_freq_table
```

index	Theme	Frequency
0	Fairs	54749
1	Amusement Parks	8237
2	Exhibitions	6342
3	Circus	2650
4	Factories	1016
5	Museums	247
6	Zoos	118
7	Seaside Resorts (Seaside Entertainment)	44
8	Road rollers	34
9	Church buildings	26

Page **1** of **2** | Go to page: 1 Show 10

Implementing Tag Recommendation Systems

In [ ]:
```
# Install the Gensim package to import Word2Vec and LDA model
!pip install gensim
```

```
Collecting gensim
  Downloading gensim-4.3.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (26.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.6/26.6 MB 54.9 MB/s eta 0:0
0:00
Collecting smart-open>=1.8.1
  Downloading smart_open-6.3.0-py3-none-any.whl (56 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.8/56.8 kB 12.9 MB/s eta 0:0
0:00
Requirement already satisfied: numpy>=1.18.5 in /opt/conda/lib/python3.9/sit
e-packages (from gensim) (1.23.5)
Requirement already satisfied: scipy>=1.7.0 in /opt/conda/lib/python3.9/site
-packages (from gensim) (1.9.1)
Installing collected packages: smart-open, gensim
Successfully installed gensim-4.3.2 smart-open-6.3.0
```

Word2Vec

In [ ]:
```
import pandas as pd
import numpy as np
from gensim.models import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity
import random
import time

# Set a random seed for reproducibility
random.seed(42)

def tag_recommendation(df, given_tag, top_N=5):
    tag_list = df['tags'].tolist()

    # Train a Word2Vec model
    model = Word2Vec(sentences=tag_list, vector_size=100, window=5, min_coun

    # Dictionary to store mappings
    tag_embedding_map = {}
    embedding_dim = 100
    # Function to get tag embedding
    def get_tag_embedding(tag, model, embedding_dim):
        if tag in model.wv:
            embedding = model.wv[tag]
            #print(f"Word Embedding of '{tag}': {embedding}")
            return embedding
        elif tag in tag_embedding_map:
            return tag_embedding_map[tag]
        else:
            # Generate a random embedding for the out-of-vocabulary tag
            random_embedding = np.random.rand(embedding_dim)
            tag_embedding_map[tag] = random_embedding
            return random_embedding
```

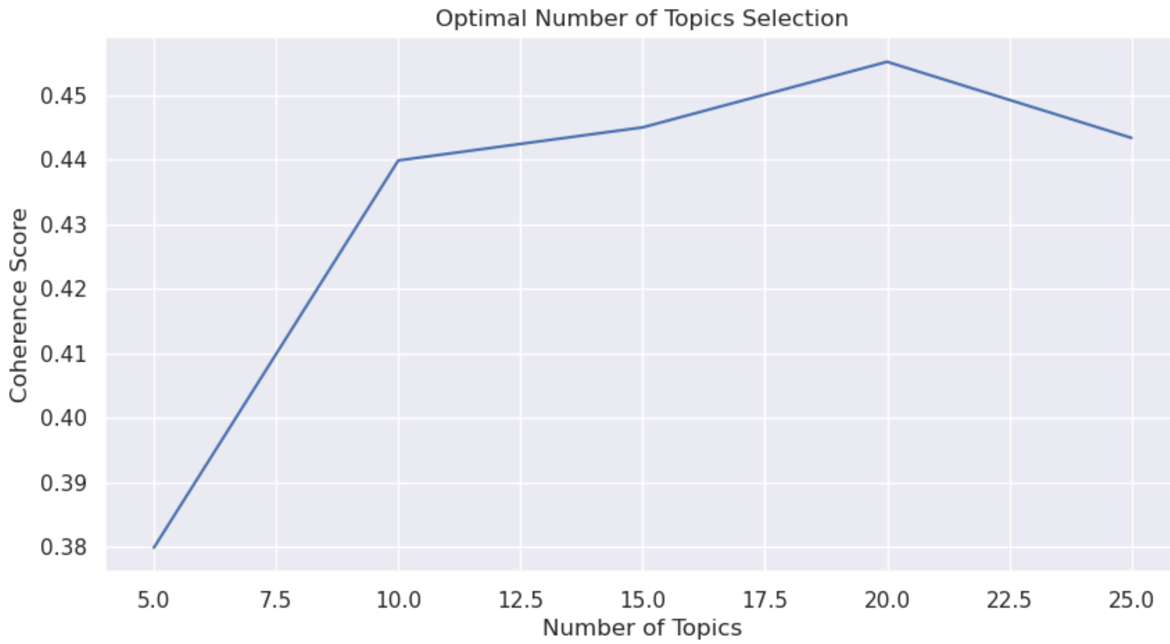
```
    # Get the word embedding for the given tag
    given_tag_embedding = get_tag_embedding(given_tag, model, embedding_dim)

    if given_tag_embedding is not None:
        # Measure the start time
        start_time = time.time()

        # Calculate the cosine similarity between the word embeddings of the
        similarity_scores = []
        unique_tokens = set(token for tokens_list in tag_list for token in t
        for tag in unique_tokens:
            tag_embedding = get_tag_embedding(tag, model, embedding_dim)
            if tag_embedding is not None:
                similarity_score = cosine_similarity([given_tag_embedding],
                if similarity_score >= 0.5:
                    similarity_scores.append((tag, similarity_score))

        # Sort the tags based on similarity scores in descending order
        similarity_scores.sort(key=lambda x: x[1], reverse=True)

        # Get the top-N recommended tags
        recommended_tags = similarity_scores[:top_N]

        # Measure the end time
        end_time = time.time()

        # Calculate the recommendation response time
        response_time = end_time - start_time

        # Get the top-N recommended tags
        total_recommended_tags = [tag for tag, _ in similarity_scores]

        # Display the total number of recommendations found
        total_recommendations = len(total_recommended_tags)
        print(f"Total Recommendations found: {total_recommendations}")

        # Display the recommendation response time
        print(f"Recommendation Response Time: {response_time:.4f} seconds")

        return  recommended_tags, total_recommended_tags ## JUST change to t
    else:
        return None

# User-defined given tag
given_tag = input("Enter your search variable : ")
recommended_tags_word2vec, total_recommendations_word2vec = tag_recommendati

# JUST comment these for overlap and Shannon’s Entropy
if recommended_tags_word2vec is not None:
    print("Recommended Tags and their Cosine Similarities:")
    for tag, similarity in recommended_tags_word2vec:
        print(f"Tag: {tag}, Cosine Similarity: {similarity:.4f}")
else:
    print("Given tag not found in the Word2Vec model.")
```

```
Total Recommendations found: 3251
Recommendation Response Time: 1.1193 seconds
Recommended Tags and their Cosine Similarities:
Tag: waltzer, Cosine Similarity: 1.0000
Tag: fogged, Cosine Similarity: 0.9758
Tag: rafter, Cosine Similarity: 0.9730
Tag: cyclone, Cosine Similarity: 0.9717
Tag: chair, Cosine Similarity: 0.9716
```

LDA

In [ ]:

```
from gensim.models import CoherenceModel
from gensim.corpora import Dictionary
from gensim.models import LdaModel
import matplotlib.pyplot as plt

# Convert ‘tags’ column to a list
tags = df['tags'].tolist()

# Create a dictionary and corpus for the LDA model
dictionary = Dictionary(tags)
corpus = [dictionary.doc2bow(doc) for doc in tags]

# Calculate coherence scores for different topic numbers
coherence_scores = []
for num_topics in range(5, 30, 5):
    lda_model = LdaModel(corpus, num_topics=num_topics, id2word=dictionary,
    coherence_model = CoherenceModel(model=lda_model, texts=tags, dictionary
    coherence_scores.append(coherence_model.get_coherence())

# Plot coherence scores
plt.plot(range(5, 30, 5), coherence_scores)
plt.xlabel("Number of Topics")
plt.ylabel("Coherence Score")
plt.title("Optimal Number of Topics Selection")
plt.show()
```



In [ ]:
```
# Install pyLDAviz package
!pip install pyLDAvis
```

```
Collecting pyLDAvis
  Downloading pyLDAvis-3.4.1-py3-none-any.whl (2.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.6/2.6 MB 55.9 MB/s eta 0:00:
00
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.9/site-packa
ges (from pyLDAvis) (3.1.2)
Requirement already satisfied: joblib>=1.2.0 in /opt/conda/lib/python3.9/sit
e-packages (from pyLDAvis) (1.2.0)
Collecting numpy>=1.24.2
  Downloading numpy-1.25.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (18.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 57.0 MB/s eta 0:0
0:00
Collecting pandas>=2.0.0
  Downloading pandas-2.0.3-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (12.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 76.3 MB/s eta 0:0
0:00
Requirement already satisfied: scipy in /opt/conda/lib/python3.9/site-packag
es (from pyLDAvis) (1.9.1)
Requirement already satisfied: gensim in /opt/conda/lib/python3.9/site-packa
ges (from pyLDAvis) (4.3.2)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.9/site-p
ackages (from pyLDAvis) (68.1.0)
Collecting funcy
  Downloading funcy-2.0-py2.py3-none-any.whl (30 kB)
Requirement already satisfied: numexpr in /opt/conda/lib/python3.9/site-pack
ages (from pyLDAvis) (2.8.3)
Requirement already satisfied: scikit-learn>=1.0.0 in /opt/conda/lib/python
3.9/site-packages (from pyLDAvis) (1.1.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/pyth
on3.9/site-packages (from pandas>=2.0.0->pyLDAvis) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.9/site
-packages (from pandas>=2.0.0->pyLDAvis) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.9/si
te-packages (from pandas>=2.0.0->pyLDAvis) (2022.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python
3.9/site-packages (from scikit-learn>=1.0.0->pyLDAvis) (3.1.0)
Collecting numpy>=1.24.2
  Downloading numpy-1.24.4-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (17.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 68.0 MB/s eta 0:0
0:00
Requirement already satisfied: smart-open>=1.8.1 in /opt/conda/lib/python3.
9/site-packages (from gensim->pyLDAvis) (6.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.9/s
ite-packages (from jinja2->pyLDAvis) (2.1.2)
Requirement already satisfied: packaging in /opt/conda/lib/python3.9/site-pa
ckages (from numexpr->pyLDAvis) (23.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.9/site-pac
kages (from python-dateutil>=2.8.2->pandas>=2.0.0->pyLDAvis) (1.16.0)
Installing collected packages: funcy, numpy, pandas, pyLDAvis
  Attempting uninstall: numpy
    Found existing installation: numpy 1.23.5
    Uninstalling numpy-1.23.5:
      Successfully uninstalled numpy-1.23.5
```

```
  Attempting uninstall: pandas
    Found existing installation: pandas 1.5.3
    Uninstalling pandas-1.5.3:
      Successfully uninstalled pandas-1.5.3
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
numba 0.56.2 requires numpy<1.24,>=1.18, but you have numpy 1.24.4 which is
incompatible.
noteable 2.0.0 requires pandas<2.0.0,>=1.5.3, but you have pandas 2.0.3 whic
h is incompatible.
fugue 0.8.3 requires pandas<2,>=1.2.0, but you have pandas 2.0.3 which is in
compatible.
flytekit 1.2.11 requires numpy<1.24.0, but you have numpy 1.24.4 which is in
compatible.
flytekit 1.2.11 requires pandas<2.0.0,>=1.0.0, but you have pandas 2.0.3 whi
ch is incompatible.
flytekit 1.2.11 requires protobuf<4,>=3.6.1, but you have protobuf 4.21.7 wh
ich is incompatible.
dx 1.3.0 requires pandas<2.0.0,>=1.3.5, but you have pandas 2.0.3 which is i
ncompatible.
databricks-sql-connector 2.5.2 requires pandas<2.0.0,>=1.2.5, but you have p
andas 2.0.3 which is incompatible.
Successfully installed funcy-2.0 numpy-1.24.4 pandas-2.0.3 pyLDAvis-3.4.1
```

In [ ]:

```python
import pyLDAvis
import pyLDAvis.gensim
# Train an LDA model
lda_model = LdaModel(corpus, num_topics=20, id2word=dictionary, passes=10)

# Prepare the visualisation
vis = pyLDAvis.gensim.prepare(lda_model, corpus, dictionary)

# Visualise the topics
pyLDAvis.enable_notebook()
pyLDAvis.display(vis)
```

Out[ ]:

In [ ]:

```python
#Implement Tag Recommendation System
import pandas as pd
from gensim.corpora import Dictionary
from gensim.models import LdaModel
from gensim.models import TfidfModel
from sklearn.metrics.pairwise import cosine_similarity
import time

def lda_tag_recommendation(df, given_tag, top_N=5):
    tags= df['tags'].tolist()

    # Create a dictionary and corpus for the LDA model
    dictionary = Dictionary(tags)
    corpus = [dictionary.doc2bow(doc) for doc in tags]

    # Train a TF-IDF model on the corpus
    tfidf = TfidfModel(corpus)
```

```
corpus_tfidf = tfidf[corpus]

# Train an LDA model
num_topics = 20
lda_model = LdaModel(corpus_tfidf, num_topics=num_topics, id2word=dictio

# Function to get topic distribution for a given document
def get_document_topics(tags):
    bow = dictionary.doc2bow(tags)
    tfidf_weights = tfidf[bow]
    topics = lda_model.get_document_topics(tfidf_weights)
    return topics

# Get topic distribution for the given tag
given_tag_topics = get_document_topics(given_tag.split())

# Get the most relevant tags for the given tag based on topic distributi
recommended_tags = []
total_recommended_tags = set()  # Use a set to avoid duplicate tags
for topic, score in sorted(given_tag_topics, key=lambda x: x[1], reverse
    topic_tags = lda_model.show_topic(topic, topn=5)  # Get the top 5 wo
    recommended_tags.extend([tag for tag, _ in topic_tags])
    total_recommended_tags.update([tag for tag, _ in topic_tags])

num_recommended_tags = len(total_recommended_tags)  # Count the number o


# Calculate recommendation response time
start_time = time.time()

# Calculate the cosine similarity between the topic distribution of the
similarity_scores = []
for tag in recommended_tags:
    tag_topics = get_document_topics(tag.split())
    given_tag_distribution = [topic for _, topic in given_tag_topics]
    tag_distribution = [topic for _, topic in tag_topics]
    cosine_sim = cosine_similarity([given_tag_distribution], [tag_distri
    similarity_scores.append((tag, cosine_sim))

similarity_scores.sort(key=lambda x: x[1], reverse=True)

# Measure the end time
end_time = time.time()

# Calculate the recommendation response time
response_time = end_time - start_time

#JUST comment these lines for overlap and Shannon’s entropy
# Display the results
print(f"Total Number of Recommendations: {num_recommended_tags}")
print(f"Recommendation Response Time: {response_time:.4f} seconds")
print("Top 5 Recommended Tags and Their Cosine Similarities:")
for tag, cosine_sim in similarity_scores[:5]:
    print(f"Tag: {tag}, Cosine Similarity: {cosine_sim:.4f}")
return total_recommended_tags  #JUST uncomment return statement for over
```

```
# User-defined given tag
given_tag = input("Enter your search variable: ")
total_recommendations_lda = lda_tag_recommendation(df, given_tag)  # JUST co
#JUST uncomment return statement for overlap and Shannon’s entropy
# recommended_tags_lda = lda_tag_recommendation(df, given_tag)
```

```
Total Number of Recommendations: 83
Recommendation Response Time: 0.0545 seconds
Top 5 Recommended Tags and Their Cosine Similarities:
Tag: bus, Cosine Similarity: 1.0000
Tag: build, Cosine Similarity: 1.0000
Tag: performing, Cosine Similarity: 1.0000
Tag: animal, Cosine Similarity: 1.0000
Tag: miscellaneous, Cosine Similarity: 1.0000
```

Overlap

In [ ]:

```
# Overlap
# Convert the lists to sets for efficient comparison
set_word2vec = set(total_recommendations_word2vec)
set_lda = set(total_recommendations_lda)

# Find the common tags by taking the intersection of the sets
common_tags = set_word2vec.intersection(set_lda)

# Count the number of common tags
num_common_tags = len(common_tags)

print(f"Number of Common Tags: {num_common_tags}")
```

```
Number of Common Tags: 75
```

Diversity

In [ ]:

```
# Shannon's entropy
import math

def calculate_entropy(recommendations):
    # Count the frequency of each recommendation
    freq = {}
    for rec in recommendations:
        if rec not in freq:
            freq[rec] = 1
        else:
            freq[rec] += 1

    # Calculate the total number of recommendations
    total = len(recommendations)

    # Calculate entropy
    entropy = 0.0
    for key in freq:
```

```
        prob = freq[key] / total
        entropy -= prob * math.log2(prob)

    return entropy
entropy_word2vec = calculate_entropy(total_recommendations_word2vec)
entropy_lda = calculate_entropy(total_recommendations_lda)

print(f"Entropy for Word2Vec Recommendations: {entropy_word2vec}")
print(f"Entropy for LDA Recommendations: {entropy_lda}")
```

```
Entropy for Word2Vec Recommendations: 11.66666784069043
Entropy for LDA Recommendations: 6.375039431346932
```